Technical Notes on Complexity of the Satisfiability Problem

Marek A. Suchenek

The Wichita State University

KS 67208 U.S.A.

Key words: algorithms' complexity, average running time, NP-complete problems, propositional calculus, P=NP problem.

AMS classification: 03B05, 68Q15.

Abstract

These notes contain, among others, a proof that the average running time of an easy solution to the satisfiability problem for propositional calculus is, under some reasonable assumptions, linear (with constant 2) in the size of the input. Moreover, some suggestions are made about criteria for tractability of complex algorithms. In particular, it is argued that the distribution of probability on the *whole* input space of an algorithm constitutes an non-negligible factor in estimating whether the algorithm is tractable or not.

1 Introduction

It is not unusual to hear computer professionals or students questioning the practical value of asymptotic complexity measures. To be honest, there is a lot of evidence of occasional discrepancy between algorithms' asymptotic and actual behaviors, for example in the area of sorting and multiplication. After all, it seems typical, that authors (like Aho, Hopcroft, and Ullman in [AHU74], end of paragraph 1.4) rather discourage the reader from drawing

too many conclusions from the fact, that a running time of an algorithm is or is not in a certain O- or Ω-class.

The theory of algorithms' complexity is logical and clear: methods used there do not seem to involve an unintentional error, since the authority of mathematics has given it its consent of the thing *quod erat demonstrandum.* So, if it is so good then why it is so bad? To investigate this paradox let us try to take a closer look at motivations of dealing with asymptotic rather than actual complexities.

There are two of them: essential machine independence of algorithms to be evaluated, and a virtual lack of limits on their inputs' size. The second implies an expectation that inputs may grow boundlessly, which is considered to be the reason for the most serious obstacle for successful termination of a run. And this is why the asymptotic complexity of an algorithm has been supposed to characterize its performance on some future inputs which (probably) may be brought to processing.

Depending on how the program's complexity is expressed in terms of its input's size, the existing asymptotic complexity measures fall into two categories: the worst-case and the average one. The first kind seems to reflect an implicit presumption of malicious gnome, who selects (possibly the most troublesome) inputs to the evaluated program. The second does not allow averaging over inputs of different sizes, which causes at least calculational problems. Both of them, apparently fully adequate for evaluation of a several program, may completely fail if applied to a comparison: program A may have overall better performance than program B, with except for a few isolated "worst" cases, when B is much quicker than A; program C may have substantially better average efficiency than program D, however only for sufficiently large inputs which may be not likely in all practical cases. Experience shows, that the above scenarios are by no means artificial.

One may suspect that dealing with future unforeseeable events may need some probability theory, and it is indeed the point of view which we advocate here. Because in estimating how much time will be spent on future computations one should take into account which elements of the input space are more likely, and which are less. Moreover, since how one measures the input size does not seem to have much influence either the on actual nor the expected

running time of any particular program, we do not see a good reason why expressing averages exclusively as a function of the size of input should be recognized as a universally satisfactory practice. (On the contrary, we have found it rather inadequate in our trials of evaluating an average running time of the programs considered in this paper). Therefore we propose a modified notions of average running time and corresponding to it O-classes.

In our approach we postpone abstraction from constant factors to some later phase of evaluation. In particular $O(\bullet^2)$ and $O(100 \times \bullet^2)$ classes are not identical in this paper. If one does not need to deal with complexity on such a concrete level, introducing appropriate equivalence relation (e.g. one can impose $O(\bullet^2) \equiv O(100 \times \bullet^2)$), will easily translate obtained results into a language of *modulo constants* complexity classes.

In the sequel, we will use an NP-complete problem, namely: the satisfiability problem of propositional calculus, as one of illustrations for our proposal. Before doing this, we will start from some theoretical considerations.

We refer the reader to any handbook on measure theory for details concerning measure and probability spaces. An extensive study of algorithms' complexity, including definitions of O- and Ω-classes, satisfiability problem, NP-completeness theory, NP-hard problems, and Cook's theorem, may be found in [PS82]. Some striking results about *better than expected* behavior of certain algorithms related to NP-hard problems may be found in [Wil84]. Shannon's counting argument in the context of complexity of Boolean functions appears in [Weg87].

2 Average running time, O-hierarchy, and tractability of algorithms

Further on, we will use the von Neuman's definition of numbers, i.e. 0 is the empty set, and $n + 1 = n \cup \{n\}$ $(= \{0, ..., n\})$. We denote the set of all numbers by ω, and the set of all of them without 0 by ω^+. Moreover, we apply $\bullet$ symbol to avoid λ - expressions. Namely, $f(\bullet)$ means $\lambda x.f(x)$, or in other words, f. E.g. $\bullet^3$ denotes the cubic function.

We will fix our attention on an algorithm P, with countable domain X (which we will call the input space), running time $T : X \to \omega^+$, distribution of probability $\mu : X \to \langle 0, 1\rangle$, extended to a normed measure $\mu : \mathcal{P}(X) \to \langle 0, 1\rangle$ by $\mu(Y) = \sum_{x\in Y} \mu(x)$ (usually, it is assumed that $\mu(X) = 1$).

By the average running time T^{μ}_{avg} (we use superscript μ to remind the explicit role of the probability distribution here) we understand a function defined for each $Y \subseteq X$ as follows:

(i) $$T^{\mu}_{avg}(Y) = \frac{\sum_{x\in Y} T(x)\times\mu(x)}{\mu(Y)}.$$

It is easily seen that the above expression defines the expected value of $T(x)$ under condition $x \in Y$, i.e. with respect to conditional probability $\mu(x)/\mu(Y)$. So its value tells us, how much time, on an average, the algorithm P will spend running on a random input x, provided it is known that $x \in Y$.

If one would like to relate a running time to the size of input, a function $f : X \to \omega$, interpreted as an input size measure, comes handy. It partitions the input space onto at most countably many non-empty subspaces, which are abstraction classes with respect to the equivalence relation $\equiv_f$ defined by: $x \equiv_f y$ iff $f(x) = f(y)$. We will use X^f_n as an abbreviation for $\{x \in X \mid f(x) = n\}$ for any $n \in \omega$. Under such conventions, a relative average running time $T^{f,\mu}_{avg}$ of P is usually defined by

(ii)
$$T^{f,\mu}_{avg}(n) = \begin{cases} 1 & \text{if } \mu(X^f_n) = 0 \\ \frac{\sum_{x\in X^f_n} T^f_x(n)\times\mu(x)}{\mu(X^f_n)} & \text{otherwise,} \end{cases}$$

where T^f_x satisfies for all $x \in X$ (or at least for those with $\mu(x) \neq 0$): $T^f_x(f(x)) = T(x)$. One can see that for each $n \in \omega$, such that $\mu(X^f_n) \neq 0$, $\sum_{x\in X^f_n} T^f_x(n) \times \mu(x) = \sum_{x\in X^f_n} T(x) \times \mu(x)$, hence

$$T^{f,\mu}_{avg}(n) = T^{\mu}_{avg}(X^f_n).$$

Unlike in the classic case, where averaging of related running time is allowed only over abstraction classes X^f_n, we admit the general case, i.e. we assume that the relative average running time may be relativized to one

partition of X, but averaged over another (one may think: orthogonal) one. However, instead of disentangling the dependency between an average value of $T(x)$ and average size of x, which does not seem simple, we will define directly, what it means that size-related average running time of P is in $O(F)$-class for some function $F : \omega \to \omega^+$. So, let $\alpha : X \to \omega$ be such a partition with corresponding abstraction classes X_n^α. We say that $T_{avg,\alpha}^{f,\mu} \in O(F)$ iff for each n, such that $\mu(X_n^\alpha) \neq 0$,

(iii) $$\textstyle\sum_{x \in X_n^\alpha} \frac{T(x)}{F(f(x))} \times \mu(x) \leq \mu(X_n^\alpha).$$

This means that the expected value of the quotient $\frac{T(x)}{F(f(x))}$ over the set X_n^α does not exceed 1.

One may check that in the usual case, where α and f coincide, $T_{avg,f}^{f,\mu} \in O(F)$ iff for each n, such that $\mu(X_n^f) \neq 0$, $T_{avg}^{\mu}(X_n^f) \leq F(n)$, that is to say, $T_{avg}^{f,\mu}(n) \leq F(n)$. Thus our definition makes a proper generalization of O-hierarchy of relative average running times.

It is not necessary that we understand f as a measure of input size. We may think of f as the running time of another program Q with input space X. In light of such an interpretation, $T_{avg,\alpha}^{f,\mu} \in O(F)$ means that F is an average upper bound of proportionality factor between the running time of P and the running time of Q, over each class X_n^α. E.g. if c is a constant then $T_{avg,\alpha}^{f,\mu} \in O(c \times \bullet)$ means that in each X_n^α, Q is on average at most c times quicker than P. If one insists on referring to the ordinary input's length, it may be measured by the time which the simple rewriting program will spent on it.

Let us remind the reader here that our intention is, at least at earlier stages of evaluation, not to abstract from the constant factor neglected in the classic definition of O-hierarchy. This is why the coefficient at $F(f(x))$ in (iii) is 1. Moreover, instead of dealing with asymptotic behavior, we purposely introduced measures for the expected behavior, which involves *all* possible inputs, so (iii) holds for all n's, not only for those greater than some n_0.

Finally, we define the notion of algorithm's tractability. We call P tractable over $Y \subseteq X$ iff

(iv) $$T^{\mu}_{avg}(Y) < \infty,$$

which means, that the expected length of the running time of P is finite, provided inputs are restricted to Y.

It follows from the above definition, that a linear algorithm (i.e. one with linear running time) may be not tractable at the same time, when an exponential one is tractable, however, for different probability distributions. To see this possibility, let $X = \omega$, $T(n) = n$, and $S(n) = 2^n$. If $\mu(n)$ is proportional to n^{-2}, and $\nu(n)$ to 2^{-2n} then $T^{\mu}_{avg}(\omega) = c \times \sum_{i\in\omega} \frac{1}{i} = \infty$ and $S^{\nu}_{avg}(\omega) = d \times \sum_{i\in\omega} 2^{-i} = 2d < \infty$.

One may notice, that in the definition of tractability, no input size measure is explicitly present. This is consistent with a simple observation that how long it takes to complete a run does not depend on how one measures the size of the corresponding input. It should be noted, however, that this natural from mathematical point of view definition may be somewhat impractical in certain cases. Clearly, if $T^{\mu}_{avg}(X) = \infty$ then you may expect the worst. But if not? Two statements $T^{\mu}_{avg}(X) < 45$ sec., and $T^{\mu}_{avg}(X) < 30{,}000$ yrs., both implying the tractability of the program in question, have quite different informational content. Because in our approach we did not abstract from constant factors while measuring program's complexity, our method may be applied as well for evaluating the tractability in a stronger sense, where, say, $T^{\mu}_{avg}(X) < 100$ hrs. is required. It is quite clear, that asymptotic complexity measures do not support, in general, this kind of estimations.

If one is interested in measuring how the actual running time is distributed around its mean, other concepts of probability theory, for instance, variance, or standard deviation, may be helpful. We will not discuss them in this paper. Let us remark, however, that since $T(x)$ is a non-negative random variable, the probability that for $x \in Y$, $T(x) \geq \alpha$ (where α is a positive constant) does not exceed $\frac{1}{\alpha} \times T^{\mu}_{avg}(Y)$. So, the computations longer than, say, $100 \times T^{\mu}_{avg}(Y)$ will occur in Y with at most 1% frequency.

For the sake of completeness of the picture we draw, let us state some basic properties relating the introduced notions to each other.

Property 2.1 If $T^{\mu}_{avg}(X) < \infty$ then for every countable partition α of input space X on subsets of positive measure,

(v) $$T^{\mu}_{avg}(X) = sup_{n\in\omega} T^{\mu}_{avg}(\cup_{i\leq n} X^{\alpha}_i).$$

Proof. By the definition, $T^{\mu}_{avg}(X) = \sum_{i\in\omega}\sum_{x\in X^{\alpha}_i} T(x)\times\mu(x)=$

$= \lim_{n\to\infty}\sum_{i\leq n}\sum_{x\in X^{\alpha}_i} T(x)\times\mu(x) = \lim_{n\to\infty}\sum_{x\in\cup_{i\leq n}X^{\alpha}_i} T(x)\times\mu(x) =$

$= \lim_{n\to\infty} T^{\mu}_{avg}(\cup_{i\leq n}X^{\alpha}_i) = ($ since $T(x)\times\mu(x)\geq 0)\ sup_{n\in\omega} T^{\mu}_{avg}(\cup_{i\leq n}X^{\alpha}_i)$. $\square$

Property 2.2 Let μ be a normed measure on input space X, let α be a countable partition of X on subsets of positive measure, let $f : x \to \omega$ be a measure of the size of input, and let $F : \omega \to \omega^+$. In such circumstances

(vi) $$T^{f,\mu}_{avg,\alpha} \in O(F)$$

iff for each distribution ν of probability satisfying

(vii) $$\nu(x) = c_H \times \frac{H(\alpha(x))}{F(f(x))} \times \mu(x),$$

where $H : \omega \to \omega$, the following inequality holds:

(viii) $$T^{\nu}_{avg}(X) \leq (F\circ f)^{\nu}_{avg}(X).$$

Proof. Let $H : \omega \to \omega$. We have:

$T^{\nu}_{avg}(X) \leq (F\circ f)^{\nu}_{avg}(X) \equiv \sum_{x\in X} T(x)\times\nu(x) \leq \sum_{x\in X} F(f(x))\times\nu(x) \equiv$

$\equiv \sum_{x\in X} T(x)\times c_H\times\frac{H(\alpha(x))}{F(f(x))}\times\mu(x) \leq \sum_{x\in X} F(f(x))\times c_H\times\frac{H(\alpha(x))}{F(f(x))}\times\mu(x) \equiv$

$\equiv \sum_{n\in\omega}\sum_{x\in X^{\alpha}_n} T(x)\times c_H\times\frac{H(\alpha(x))}{F(f(x))}\times\mu(x) \leq \sum_{n\in X} c_H\times H(\alpha(x))\times\mu(x) \equiv$

(ix) $$\equiv \sum_{n\in\omega} H(n)\times\sum_{x\in X^{\alpha}_n}\frac{T(x)}{F(f(x))}\times\mu(x) \leq \sum_{n\in\omega} H(n)\times\mu(X^{\alpha}_n).$$

If (vi) is true then by (iii) and (ix), we get (viii).

For proof of the converse implication let us assume (viii) and take as H in (vii) the characteristic function of the set $\{m\}$, where $m \in \omega$. In this case (ix) may be reduced to

$\sum_{x \in X_m^\alpha} \frac{T(x)}{F(f(x))} \times \mu(x) \leq \mu(X_m^\alpha)$, which gives (vi). □

Let us note here that constant c_H in (vii) is unambiguously determined by H, since $\nu(X) = 1$. Moreover, if $f = \alpha$ then $F(f(x))$ in (vii) may be omitted.

Property 2.3 Let α be a countable partition of input space X, let $f : x \to \omega$ be a measure of the size of input, let μ be a measure normed on each X_n^α (i.e. $\mu(X_n^\alpha) = 1$ for all $n \in \omega$), and let $F, H : \omega \to \omega$. If for each $n \in \omega$

(x) $$T_{avg,\alpha}^{f,\mu} \in O(F)$$

then for every distribution ν of probability satisfying

(xi) $$\nu(x) \leq \frac{H(\alpha(x))}{F(f(x))} \times \mu(x)$$

the following implication holds :

(xii) $$\sum_{n \in \omega} H(n) < \infty \supset T_{avg}^{\nu}(X) < \infty.$$

Proof. (x) means that for each $n \in \omega$:

(xiii) $$\sum_{x \in X_n^\alpha} \frac{T(x)}{F(f(x))} \times \mu(x) \leq 1.$$

Hence $T_{avg}^{\nu}(X) =$ (by (i) and $\nu(X) = 1$) $\sum_{x \in X} T(x) \times \nu(x) \leq$

$= \sum_{n \in \omega} (\sum_{x \in X_n^\alpha} T(x) \times \frac{\mu(x) \times H(\alpha(x))}{F(f(x))}) = \sum_{n \in \omega} (H(n) \times \sum_{x \in X_n^\alpha} \frac{T(x)}{F(f(x))} \times \mu(x)) \leq$ (by xiii)

$\leq \sum_{n \in \omega} H(n)$, that is to say, $T_{avg}^{\nu}(x) \leq \sum_{n \in \omega} H(n)$, which gives us (xii). □

The above properties are useful in estimating tractability of algorithms. Property 2.1 gives us a tool for direct calculations of $T^{\mu}_{avg}(X)$. Using it one may also investigate the rate of growth of T^{μ}_{avg} in function of $\cup_{i<n} X^{\alpha}_{i}$, which may be useful if $T^{\mu}_{avg}(X)$ is infinite, or finite but prohibitively large. Putting $\alpha = f$ one can use known facts about average running time in classic sense in estimating the tractability. However, it may be somewhat difficult to discover a useful formula describing $T^{\mu}_{avg}(\cup_{i<n} X^{f}_{n})$. Property 2.2 allows estimations of tractability in all cases the behavior of F of is known. Property 2.3 (being as a matter of fact a generalization of Property 2.1) may prove suitable in cases Property 2.1 is not. It allows local analysis (i.e. in X^{α}_{n} subspaces) which using this property may be extended to the whole input space.

3 Complexity of tabulating program

As the first example of application of the introduced notions, let us evaluate the complexity of a program, which given a sentence of propositional calculus tabulates the Boolean function defined by that sentence. The problem of such tabulation is NP-hard.

Even relatively simple algorithms (as one rewriting input to output) may be intractable if the distribution of probability does not decrease fast enough with the growth of input size. Therefore to have a tractable instance of the problem one has to impose some conditions on rate of fading of probability distribution. Surprisingly, a relatively modest condition will suffice for this end.

We will start from input space X containing binary representations (using e.g. ASCII or EBCDIC codes) of all propositional sentences in the reverse Polish form, which are composed of some countably infinite set of propositional variables, and any complete set of logical connectives (e.g. $\vee, \wedge$, and $\neg$). As input size measure f we will adopt the length (in bits) of the representation mentioned above. As the orthogonal partition α we will use the number $\alpha(x)$ of propositional variables appearing in the input x (α and f are not fully independent, since $f(x)$ cannot be less than $\alpha(x)$; we will not use this fact, however). We will assume, that the running time of the program for any input x is equal to $2^{\alpha(x)} \times f(x)$, measured in some abstract

units of time. It is quite obvious, that there exists an algorithm returning this “efficiency”: if it runs too fast, it delays in printing the answer until the time $2^{\alpha(x)} \times f(x)$ will have been exhausted. Of course, one can probably construct a faster program, but this one will suffice for our purposes. It is perhaps paradoxical, nevertheless clear, that only the tiny inputs are causing problems with relative efficiency of our algorithm, since for large inputs x of size greater than $2^{\alpha(x)}$ it has quite good, linear performance. On the other hand, the number of such tiny inputs is relatively so small in comparison to the number of all non-equivalent propositions of minimal lengths that it may be unable to lead us away from polynomial average hierarchy.

We will split each X^α_n (the set of all propositions of X with n propositional variables) onto a family of its subsets $Y^n_0, Y^n_1, ..., Y^n_i, ...$, so that Y^n_0 will consist of some sort of shortest sentences of X^α_n, Y^n_1 of the same sort of sentences of $X^\alpha_n \setminus Y_0$, and so on. Namely, we define a function $min : \mathcal{P}(X) \to \mathcal{P}(X)$ by:

(i) for every element of $Y \subseteq X$ there exists a logically equivalent to it element of $min(Y)$

(ii) for every element x of $min(Y)$ and every element y of Y, if x is logically equivalent to y then $f(x) \leq f(y)$

(iii) $min(Y)$ is a minimal set satisfying (i) and (ii).

To demonstrate the existence of such $min(Y)$ one has to make use of the axiom of choice: from each class of abstraction for the logical equivalence on Y pick up an element x with possibly smallest value of $f(x)$. The set constructed this way happened to automatically satisfy condition (iii).

Now for each $n \in \omega$ pose $Y^n_0 = min(X^\alpha_n)$, and $Y^n_{i+1} = min(X^\alpha_n \setminus \cup_{j \leq i} Y^n_j)$. Of course, we have

(iv) $$X^\alpha_n = \cup_{i \in \omega} Y^n_i,$$

and for any $i \neq j$, $Y^n_i \cap Y^n_j = 0$. Let us estimate lower bounds for lengths of codes of elements in Y^n_i. Each Y^n_i contains the number of elements equal to the cardinality of Lindenbaum’s algebra with n generators, or - equivalently - of Boolean algebra of functions with n variables, that is to say, 2^{2^n}. Let us assume, that probability distribution μ assigns the same value to all elements of Y^n_i. A semantical argument of 1-1 correspondence between the elements

of Y_i^n and elements of mentioned above algebras shows, that this assumption is reasonable. It will enable us to apply Shannon's counting argument.

To evaluate the value of $\sum_{x\in Y_i^n} \frac{T(x)}{f^3(x)} \times \mu(x)$, equal to $\sum_{x\in Y_i^n} \frac{2^n}{f^2(x)} \times \mu(x)$, let us observe that for every function $g : \omega \to \omega$ such that for all x, $g(x) \leq f(x)$, the inequality $\sum_{x\in Y_i^n} \frac{2^n}{f^2(x)} \times \mu(x) \leq \sum_{x\in Y_i^n} \frac{2^n}{g^2(x)} \times \mu(x)$ holds. Therefore we may safely assume that each Y_i^n contains all 2^{2^n} shortest binary codes, giving the absolute lower bound of f for all Y_i^n together. In this case Y_i^n is composed of all the codes of length $\leq 2^n - 1$, and of one code of length 2^n.

We have:

$\sum_{x\in Y_i^n} \frac{T(x)}{f^3(x)} \times \mu(x) = \sum_{x\in Y_i^n} \frac{2^n}{f^2(x)} \times \mu(x) = 2^n \times \mu(y_0) \times \sum_{x\in Y_i^n} \frac{1}{f^2(x)} =$

$= 2^n \times \mu(y_0) \times (\sum_{i=1}^{2^n-1} \frac{1}{i^2} \times 2^i + \frac{1}{(2^n)^2}) \leq 2^n \times \mu(y_0) \times \sum_{i=1}^{2^n} \frac{1}{i^2} \times 2^i \leq 2^n \times \mu(y_0) \times 2^n \times \frac{1}{(2^n)^2} \times 2^{2^n} =$

$= \mu(y_0) \times 2^{2^n} = \mu(Y_i^n)$, where y_0 is any element of Y_i^n. From (iv) follows

$\sum_{x\in X_n^\alpha} \frac{T(x)}{f^3(x)} \times \mu(x) = \sum_{i\in\omega} \sum_{x\in Y_i^n} \frac{T(x)}{f^3(x)} \times \mu(x) \leq \sum_{i\in\omega} \mu(Y_i^n) = \mu(\cup_{i\in\omega} Y_i^n) = \mu(X_n^\alpha)$.

According to our definition of O-class, it means that $T_{avg,\alpha}^{f,\mu} \in O(\bullet^3)$. Applying Property 2.3 and taking into account $\alpha(x) \leq f(x)$ we conclude that if for every x,

$\nu(x) \leq c \times \frac{\mu(x)}{f^d(x)\times\mu(X_n^\alpha)}$, where $d > 4$, then $T_{avg}^\nu(X) < \infty$.

We were not able to draw this conclusion using exclusively Property 2.1, which suggests that our generalized notion of average running time O-hierarchy may be more useful that the classic one.

4 Complexity of the satisfiability problem

The satisfiability problem of propositional calculus may be formulated as follows.

> Given a sentence φ of propositional calculus, decide whether there exists a truth-valued assignment for its proportional variables making φ true.

All known deterministic solutions to the satisfiability problem are of exponential worst-case time complexity. However, the question of existence of polynomial solution still remains open. If the answer is "yes" then, as Cook has shown, every problem, which may be non-deterministically solved in polynomial worst-case time, can also be solved deterministically in polynomial worst-case time. This is the celebrated P=NP problem.

Instead of investigating the worst-case running time of the quickest solution of the satisfiability problem, we will answer more practical question of its tractability, instead. A positive result we have been able to achieve in this respect makes, in our opinion, the P=NP problem slightly less dramatical.

One may expect, that testing the satisfiability should be easier than tabulating a Boolean function. Indeed, for all but unsatisfiable sentences (describing the constant false Boolean function) one may stop trying all possible assignments after the first satisfying one has been found. Now our program will stop either if it found an assignment making its input sentence true or if it examined unsuccessfully all possible assignments.

How much time will it save us on average? We will show that surprisingly much, as it follows from an elementary property of subsets of the set $M = \{0, ..., M-1\}$: assuming fair distribution of probability on $\mathcal{P}(M)$, the expected value of minimal element in a random subset of M (which is the same as the expected number of tosses of a coin until *heads* appears) is less than 2, no matter how large is M. Qualitatively similar observation one can find in [Wil84], pages 216–221, where the author proves that the average number N of nodes in the backtrack search tree of a *random* graph subjected to coloring with at most n colors may be approximated regardless of the size of the graph; e.g. if $n = 3$ then $N \approx 197$.

With each proposition φ_n of n proportional variables we will associate its model: a set $\mathcal{K}(\varphi_n)$ of all assignments, coded as binary sequences of length n, which make φ_n true. Since every such sequence constitutes a number from the interval $\langle 0, 2^n - 1\rangle$, models may be thus understood as subsets

of $2^n = \{0, ..., 2^n - 1\}$. We assume that the program testing satisfiability scans all numbers m from 0 to 2^{n-1}, verifying for each m, whether its binary representation satisfies a sentence in question or not.

The time (measured in some abstract units) our program will spent on any input x with n propositional variables is given by:

$$T(x) = f(x) \times (min_n(\mathcal{K}(x)) + 1)$$

where

$$min_n(\mathcal{K}) = \begin{cases} 2^n & \text{iff } \mathcal{K} = 0 \\ min(\mathcal{K}) & \text{otherwise} \end{cases}$$

Having a model $\mathcal{K}$ one may think of the set of all propositions φ_n, for which $\mathcal{K}$ is the model. Let us denote it by $Th(\mathcal{K})$. Using similar semantical argument as in section 3, we assume that given n, it is equally likely that a random formula φ falls in any class $Th(\mathcal{K})$. In terms of probability distribution μ it means that for each n and every two $\mathcal{K}, \mathcal{L} \subseteq 2^n, \mu(X_n^\alpha \cap Th(\mathcal{K})) = \mu(X_n^\alpha \cap Th(\mathcal{L}))$.

We have:

$$\sum_{x \in X_n^\alpha} \frac{T(x)}{2 \times f(x)} \times \mu(x) = \sum_{\mathcal{K} \subseteq 2^n} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \frac{T(x)}{2 \times f(x)} \times \mu(x) =$$

$$= \sum_{\mathcal{K} \subseteq 2^n} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \frac{min_n(\mathcal{K})+1}{2} \times \mu(x) = \sum_{\mathcal{K} \subseteq 2^n} \frac{min_n(\mathcal{K})+1}{2} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \mu(x) =$$

$$= \sum_{\mathcal{K} \subseteq 2^n} \frac{min(\mathcal{K})+1}{2} \times \mu(X_n^\alpha \cap Th(\mathcal{K})) = \sum_{\mathcal{K} \subseteq 2^n} \frac{min(\mathcal{K})+1}{2} \times \frac{\mu(X_n^\alpha)}{2^{2^n}} =$$

$$= \frac{\mu(X_n^\alpha)}{2} \times \sum_{\mathcal{K} \subseteq 2^n} \frac{min_n(\mathcal{K})+1}{2^{2^n}} = \frac{\mu(X_n^\alpha)}{2} \times \sum_{i=1}^{2^n} \frac{i \times 2^{2^n - i}}{2^{2^n}} < \frac{\mu(X_n^\alpha)}{2} \times \sum_{i=1}^{\infty} \frac{i}{2^i} = \mu(X_n^\alpha).$$

Hence $T_{avg,\alpha}^{f,\mu} \in O(2 \times \bullet)$.

Applying Property 2.3 we conclude that if for every x,
$\nu(x) \leq c \times \frac{\mu(x)}{f^d(x) \times \mu(X_n^\alpha)}$, where $d > 2$, then $T_{avg}^\nu(X) < \infty$. Again we were not lucky enough to get the same result using classic complexity measures.

The same calculations prove the above for the co-problem. Also, the NP-completeness of the satisfiability problem seems to be a rich source of similar estimations for other known complex problems. E.g. the mentioned above graph coloring with backtrack search, or simplex algorithm (see [Wil84] for its analysis) have been known to have better than exponential average performance.

5 Higher order moments

Similar calculations show that the m-th moment of $T(x)$, that is to say, the average $m-$th power of the running time of the program mentioned in section 4 is in
$O(2.5 \times m^{m+1} \times \bullet^m)$. Namely, for $m \geq 2$ we have:

$\sum_{x \in X_n^\alpha} \frac{T^m(x)}{2.5 \times m^{m+1} \times f^m(x)} \times \mu(x) = \sum_{\mathcal{K} \subseteq 2^n} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \frac{T^m(x)}{2.5 \times m^{m+1} \times f^m(x)} \times \mu(x) =$

$= \sum_{\mathcal{K} \subseteq 2^n} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \frac{(min_n(\mathcal{K})+1)^m}{2.5 \times m^{m+1}} \times \mu(x) = \sum_{\mathcal{K} \subseteq 2^n} \frac{(min_n(\mathcal{K})+1)^m}{2.5 \times m^{m+1}} \sum_{x \in X_n^\alpha \cap Th(\mathcal{K})} \mu(x)$
$=$

$= \sum_{\mathcal{K} \subseteq 2^n} \frac{(min(\mathcal{K})+1)^m}{2.5 \times m^m} \times \mu(X_n^\alpha \cap Th(\mathcal{K})) = \sum_{\mathcal{K} \subseteq 2^n} \frac{(min(\mathcal{K})+1)^m}{2.5 \times m^{m+1}} \times \frac{\mu(X_n^\alpha)}{2^{2^n}} =$

$= \frac{\mu(X_n^\alpha)}{2.5 \times m^{m+1}} \times \sum_{\mathcal{K} \subseteq 2^n} \frac{(min_n(\mathcal{K})+1)}{2^{2^n}} = \frac{\mu(X_n^\alpha)}{2.5 \times m^{m+1}} \times \sum_{i=1}^{2^n} \frac{i^m \times 2^{2^n - i}}{2^{2^n}} \leq \frac{\mu(X_n^\alpha)}{2.5 \times m^{m+1}} \times$
$\sum_{i=1}^{\infty} \frac{i^m}{2^i}$.

On the other hand, $\sum_{i=1}^{\infty} \frac{i^m}{2^i} = \sum_{i=1}^{m} \frac{i^m}{2^i} + \sum_{i=m+1}^{\infty} \frac{i^m}{2^i} \leq \sum_{i=1}^{m} \frac{m^m}{2^i} + \sum_{i=m+1}^{\infty} (\frac{i}{2^{\frac{i}{m}}})^m =$

$= m^m \times \sum_{i=1}^{m} 2^{-i} + \sum_{\xi = \frac{m+1}{m}, \Delta\xi = \frac{1}{m}}^{\infty} (\frac{\xi \times m}{2^\xi})^m \leq m^m + m^m \times \sum_{\xi = \frac{m+1}{m}, \Delta\xi = \frac{1}{m}}^{\infty} (\frac{\xi}{2^\xi})^m \leq$

$\leq m^m \times (1 + m \times \sum_{k=1}^{\infty} (\frac{k}{2^k})^m \leq m^m \times (1 + m \times \sum_{k=1}^{\infty} \frac{k}{2^k} \leq \frac{1}{2} \times m^{m+1} + 2 \times m^{m+1} \leq$

$2.5 \times m^{m+1}$.

Hence $\sum_{x \in X_n^\alpha} \frac{T^m(x)}{2.5 \times m^{m+1} \times f^m(x)} \times \mu(x) \leq \mu(X_n^\alpha)$, i.e. $(T^m)_{avg,\alpha}^{f,\mu} \in O(2.5 \times m^{m+1} \times \bullet^m)$.

There is a surprising (please take into account approximate calculations) coincidence between the constant 197 for 3-coloring backtrack search of [Wil84], page 216, and the constant $3^3 + 2 \times 3^{3+1} = 189$ of our estimation.

6 A grain of salt

As we have seen in two previous sections, under rather acceptable assumptions we calculated that the expected running time of tabulating algorithm does not exceed the cube of the time needed for merely rewriting the input, and that the expected running time of satisfiability testing is less than three times greater than the time spent on reading the input. Those result may or may not hold for other probability distributions. Despite its seemingly naturalness, the assumption of section 3 we have made about $\mu(Y_i^n)$ is rather strong; as a matter of fact, it implies that the probability of a sentence decreases exponentially with the number of distinct variables it contains. (Here Shannon's counting argument fights back). In our opinion it cannot be precluded that it is the most likely probability distribution in Artificial Intelligence applications, where verified sentences are rather far from being random in a lexical sense. However, if we assume, that the probability μ decreases with p-th power of input's length then the following example shows that $T_{avg}^{\mu}(X) = \infty$.

Example 6.1 Consider a language containing all and only 16 binary connectives (i.e. names of binary Boolean functions). Elementary calculations show that there are

$$\Gamma(N) \times 16^N \times (2^{N+1} - 1)$$

different sentences containing exactly N connectives (and therefore $N + 1$ propositional variables; the set V of this variables we treat as fixed here), where $\Gamma(N)$ is defined inductively:

$\Gamma(0) = 1$,

$\Gamma(n+1) = \sum_{i=0}^{n} \Gamma(i) \times \Gamma(n-i)$,

and denotes the number of different types of sentences one may construct out of N binary connectives. Factor $2^{N+1}-1$ is the number of possible selections from V.

Total time of reading all these sentences is equal to $(2N+1) \times \Gamma(N) \times 16^N (2^{N+1}-1)$, while total time of their tabulating is $(2N+1) \times \Gamma(N) \times 16^N \times (3^{N+1}-1)$. Therefore the ratio $F(N) = \frac{3^{N+1}-1}{2^{N+1}-1} \times (2N+1)^{-p} \approx (1.5)^{N+1} \times (2N+1)^{-p}$ cannot have the convergent sum, i.e. $\sum_{N=0}^{\infty} F(N) = \infty$.

The same is true if we assume, that input's probability decreases with p-th power of the number of its propositional variables. $\square$

The situation becomes diametrically different if one assumes to have in the language all possible n-ary connectives for each $n < 0$, with fair distribution of probability over arity classes. This means that each n-ary Boolean function has in this language its individual name which may appear in input equally likely with any other name of n-ary Boolean function. The explosion of connectives and lengths of their codes should substantially contribute to the enhancement of average relative running time of tabulating program: one may easily verify than assumption that μ is constant on Y_i^n is satisfied in this case.

The situation with the satisfiability problem is, hopefully, not as clear, because we did not use Shannon's counting argument here. Of course, having all possible and equally likely connectives in a language forces that the assumption of $\mu(X_n^\alpha \cap Th(\mathcal{K})) = \mu(X_n^\alpha \cap Th(\mathcal{L}))$ is met. The more problematic case, where, say, the arity of connectives is bounded, e.g. it cannot exceed 2, requires further investigation. The answer to this problem is, probably, hidden in the following question:

> Assuming that all and only N-ary connectives are present in the object language, and that any two sentences of the same length have the same probability, given number M, what is the expected value of $min_{\alpha(x)}(\mathcal{K}(x))$, where x is a random element of X_M^f?

7 A comparison of methods

In our opinion, the expected complexity $T^{\mu}_{avg}(X)$, and in particular its finiteness, is the most adequate complexity measure, provided P is intended for frequent future use, and the distribution of probability μ really describes what is going on in its input. The role of other characteristics, like T^f_x, $T^{f,\mu}_{avg}$, or $T^{f,\mu}_{avg,\alpha}$, as well as asymptotic measures of complexity, is secondary, as they serve as a calculational facility in estimating the value of $T^{\mu}_{avg}(X)$. Incidentally, the knowledge of worst-case or average running time in the classic sense, or at least some O-class to which it belongs, may be sufficient to prove that $T^{\mu}_{avg}(X) < \infty$, using e.g. Property 2.1, but, as we have seen, not necessarily in all cases. On the other hand, a peculiar conviction that $\mathrm{O}(\bullet)$ is much better than $\mathrm{O}(2^{\bullet})$ in circumstances when the probability that in the next run the input will have given length decreases with its second power, seems like preferring rain to mud: both of them cause nontractability problems.

If one insists on having a characterization of how an increase in size of input space would affect the tractability of an algorithm, Property 2.1 is a neat tool for the purpose. It may be useful, e.g,. for finding a maximal N such that $T^{\mu}_{avg}(\cup_{i\leq N} X^{\alpha}_i) \leq c$, where c is a limit of one's average patience. Since, on general, values of $T^{\mu}_{avg}(\cup_{i\leq N} X^{f}_i)$ and $T^{\mu}_{avg}(X^f_N)$ may differ from each other considerably, using to this end the classical concept of average running time, besides some unnecessary calculational problems which result from restricting α to f, may lead to false conclusions. Obviously, asymptotic measures may be impractical in such a case, since N we are interested in may be not large enough, i.e. less than n_0 appearing in the definition of O-class.

Asymptotic measures may be adequate iff the probability of inputs of some small size is appropriately small, which would probably happen in most cases where probabilities of any two inputs, or at least of any two input's lengths, were the same. However, if the input space is infinite, then such distribution of probability is impossible, since in this case

$$\mu(X) = \sum_{x\in X} \mu(x) = \begin{cases} \sum_{x\in X} 0 = 0 \neq 1, & \text{if } \mu(x) = 0 \\ \sum_{x\in X} \varepsilon = \infty \neq 1, & \text{otherwise.} \end{cases}$$

In our opinion the above fact is one of the reasons for discrepancies between asymptotic and actual behaviors of many algorithms.

Using a worst-case measure in estimating algorithm efficiency is equivalent to average case if the probability of non-worst inputs vanishes. This is true under, as we call it, the malicious gnome assumption.

8 Final remarks

Many people are quite skeptical about adequacy of probability theory, seemingly expecting somebody to demonstrate the "truthfulness" of its axioms. We do not share their reservations, consciously leaving the choice of pertinent probability measure to lucky guessing of the applier. It does not mean, however, that we see the results obtained on the ground of this theory as nothing but speculations. In particular, we have found it a little bit surprising, nevertheless instructive, that under quite realistic assumptions a simple reading program may need, on average, as much as 30 % of the running time of a satisfiability checker. This is why we wrote this paper.

References

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*, pages 124–145. Addison-Wesley Publishing, 1974.

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc, 1982.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions*. B. G. Teubner, and John Wiley & Sons, 1987.

[Wil84] S. Wilf. Backtrack: An O(1) expected time algorithm for the graph coloring problem. In *In Information Processing Letters*, volume 18, 1984.